Cséri Tamás

Információnyerés C++ programokból

Kutatási terv

ELTE Informatikai Kar Doktori Iskola

Témavezető: Dr. Porkoláb Zoltán

1. Témaválasztás

Az információ visszanyerése korábban megírt programokból létszükségletű a programok megértése, tesztelése, karbantartása, továbbfejlesztése céljából. Ez a technológia fontos a modell-alapú fejlesztés, az intencionális programozás, és a kód vizualizáció témakörében is. Ebben a kutatási témakörben az ELTE Informatikai Karán egy új, nagy jelentőségű kutatás indult az Ericsson Magyarország megbízásából. A kutatás célja, hogy létező nagyméretű C++ források statikus elemzése segítségével visszaállítsa a program egy részletes modelljét. Ennek a modellnek a segítségével lehetővé válik a modell megjelenítése, elemzése és a forráshoz kapcsolódó számos egyéb alkalmazás létrehozása. Ezek az alkalmazások kapcsolatot teremthetnek olyan rendszerekkel, mint a verziókezelő rendszerek, teszt-eszközök, szoftver-metrikák.

E projekt során a forráskód parszolása segítségével létre kell hozni a program belső reprezentációját, majd az így kapott struktúrán további feldolgozásokat kell végezni. Az így kapott szemantikus gráf lesz a keresett modell alapja, erre a struktúrára épülnek a további feldolgozások. Távlati tervek között szerepelhetnek különböző programtranszformációk, kódlefedettségi és kódszelekciós módszerek, végrehajtási időt figyelő számítások stb.

2. Kapcsolódó kutatások

A programok statikus elemzése nagy múltra visszatekintő kutatási téma [1]. Ugyanakkor, specifikusan a C++ nyelvre [2,3] viszonylag kevés eredmény született eddig. Ennek oka részben a nyelv bonyolult szintaxisa, részben pedig a nyelv multiparadigmás jellege. Nem elhanyagolható problémát jelent a preprocesszor, mely a nyelv szabványos része, ezáltal egyfajta kétszintű nyelvet létrehozva a C++-ból. Hasonlóan kritikus a template-ek kezelése is. Emiatt számos kutató valamely egyszerűbb nyelvet választ kutatási célul.

Jelentős kutatások születtek a C++ nyelv statikus elemzésekor a slicing témakörében. Slicing alatt azt értjük, hogy kiválasztjuk a forráskód azon részét, amely hatással van valamely kódrészlet (tipikusan egy változó) értékére [4]. Statikus slicing [5] esetében mindezt forráskódból, dinamikus slicing [6] esetében valamely program lefutás során történő

vizsgálatból próbáljuk ezt megtenni. Legújabb kutatások már foglalkoznak a preprocesszor és a slicing közös kérdéseivel [7].

Komoly C++-szal kapcsolatos kutatások foglalkoznak a Feature-alapú programozás keretein belül, illetve a Software Product Line elvei szerint. Ezek a módszerek abból indulnak ki, hogy a legtöbb mai nagy szoftver számos ki-be kapcsolható részfunkcióval (feature) rendelkezik. Ezeket a funkciókat, valamint a legkülönbözőbb konfigurációs paramétereket elemezve számos potenciális hibát ki lehet szűrni [8,9].

Statikus elemzés segítségével, a legutóbbi kódváltozások kinyerésével megállapítható, hogy a program mely funkciói változtak. Ezáltal csökkenthető az újra lefuttatandó tesztesetek száma. Ez az ún. teszt-kiválasztás is fontos kutatás, mivel a nagy rendszerek regressziós tesztjeinek teljes újrafuttatása rendkívül idő- és erőforrás igényes feladat [10,11]

A program modell-alapú ábrázolására és fejlesztésére törekszik a szándékalapú programozás, amit Charles Simonyi szorgalmaz [12]. A szándékalapú programozás elsősorban új kód létrehozására szolgál, de természetesen szerves része meglévő programok beolvasása, és modell-alapúvá konvertálása.

A programok statikus analízise, és modell-szerű ábrázolása az egyik legelterjedtebb megoldás refaktorálás során. C++ nyelvre viszonylag kevés eredmény született e témakörben, a legjelentősebb és egyik legtöbbször hivatkozott cikk [13], mely részletesen leírja, hogyan kell kezelni a preprocesszor direktívákat refaktorálás során.

Nem C++ nyelvi eredmény, de a kutatás során mindenképpen fontos kiindulópont az ELTE Informatikai Kar Szoftverlaboratóriumának Erlang refaktorálási kutatása [14], mely a mi kutatásunkhoz hasonló modellt épít fel. Ugyancsak itt definiáltak egy ERLANG szemantikus lekérdező nyelvet [15], amihez hasonlót szeretnénk C++-hoz készíteni.

Az IBM ugyancsak foglalkozik C++ forráskódok elemzésével. Az általuk grokking-nak nevezett megértési feladat során a forráskódból bizonyos mintákat próbálnak azonosítani, akár félautomatikusan, azaz szakértői segítséget is igénybe véve [16].

3. Tervezett kutatási irány

A kutatás során elsőként definiálni kell azt az absztrakt modellt, ami képes információveszteség nélkül tárolni C++ programokat. A modell teljes kell legyen abban az értelemben, hogy belőle nem csak az ekvivalens eredményt adó programot kell tudni visszanyerni, de a visszagenerált kódnak a lehető legnagyobb formális azonosságban kell állnia az eredetivel. Így lehetőség szerint a modellnek komment-, whitespace- és stílus-tartónak kell lennie.

Másodszorra meg kell valósítani a forráskód parszolását, a modell előállítását. Erre a célra valamely nyílt forráskódú szabad felhasználású szoftver tűnik a legalkalmasabbnak, pl. a clang [15].

Ezek után az elkészült modellben tárolt információk visszanyerésére egy speciális szemantikus lekérdezőnyelvet szeretnénk definiálni. Ilyen nyelv már létezik ERLANG nyelvre [13], ezt kell általánosítani, hogy a multiparadigmás C++ nyelv számára is megfelelő legyen.

A gyakran alkalmazott programozási konstrukciókat tervmintáknak nevezzük. Egyes tervmintákat a programozók ismernek, így ezek felismerése lehetővé teszi a kód gyorsabb megértését. Olyan tervminták is léteznek, melyek előfordulása azt jelenti, hogy a programunk nem jó (antiminták). Ezek felismerése lehetővé teszi a programozó számára, hogy elkerülje e konstrukciók használatát, ezáltal növeli a kód minőségét. Az elkészült modellt dekorálni kell domain-specifikus tervmintáknak megfelelő információkkal. Ezeket az információkat – domain pattern-eket – egy külső eszközzel adja meg az adott szakterület specialistája. Kérdés, hogy milyen módon ábrázoljuk ezt a tudást. Lehetőség szerint a már említett szemantikus lekérdező nyelvet alkalmaznánk ebből a célból.

4. Hivatkozások

[1] R. Ferenc, A. Beszedes and T. Gyimothy. *Extracting Facts with Columbus from C++ Code*.In Tool Demonstrations of the 8th European Conference on Software Maintenance and Reengineering (CSMR 2004), Tampere, Finland, pages 4–8, March 24-26, 2004.

[2] Bjarne Stroustrup. *The Design and Evolution of C++*, Addison-Wesley Professional, 1st edition, 1994. ISBN 0201543303.

[3] Bjarne Stroustrup. *The C++ Programming Language, Special Edition*. Addison-Wesley Professional, 3rd, special edition, 2000. ISBN 0201700735.

[4] F. Tip, *A survey of program slicing techniques*. Journal of Programming Languages, 3(3):121-189, Sept. 1995.

[5] Jens Krinke. 1998. *Static slicing of threaded programs*. In Proceedings of the 1998 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (PASTE '98), A. Michael Berman (Ed.). ACM, New York, NY, USA, 35–42.

[6] A. Beszedes, T. Gergely, Zs. M. Szabo, J. Csirik, T. Gyimothy. *Dynamic slicing method for maintenance of large C programs*, CSMR 2001, pages 105–113.

[7] László Vidács, Árpád Beszédes, Tibor Gyimóthy. *Combining preprocessor slicing with C/C++ language slicing*, Sci. Comput. Program. 74, 7 (May 2009), 399–413.

[8] Christian Kästner, Paolo G. Giarrusso, Klaus Ostermann. *Partial preprocessing C code for variability analysis*, In Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, p.127–136, January 27-29, 2011, Namur, Belgium.

[9] Andy Kenner, Christian Kästner, Steffen Haase, Thomas Leich. *TypeChef: toward type checking* `#ifdef` *variability in C*, In Proceedings of the 2nd International Workshop on Feature-Oriented Software Development, p.25–32, October 10-10, 2010, Eindhoven, The Netherlands.

[10] G. Rothermel and M. Harrold. *Analyzing regression test selection techniques*. IEEE Transactions on Software Engineering, 22(8):529--551, August 1996.

[11] G. Rothermel, M. J. Harrold, and J. Dedhia. *Regression test selection for C++ software*. J. Softw. Testing, Verif., and Rel., 10(2), June 2000.

[12] Charles Simonyi, Magnus Christerson, and Shane Clifford. 2006. *Intentional software*. InProceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA '06). ACM, New York, NY, USA, 451–464.

[13] Alejandra Garrido. *Program refactoring in the presence of preprocessor directives*, Doctoral Dissertation, University of Illinois at Urbana-Champaign Champaign, IL, USA 2005, ISBN:0-542-44639-1, Advisor: Ralph Johnson.

[14] Tóth, M., Bozó, I., Horváth, Z., Erdődi, A. *Static analysis and refactoring towards Erlang multicore programming*. Pre-proceedings of the Fourth Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, PLACES'11

[15] Lövei, L., Hajós, L., and Tóth, M. *Erlang Semantic Query Language*, 8th International Conference on Applied Informatics, ICAI 2010. Eger, Hungary, January 2010.

[16] Maayan Goldstein and Dany Moshkovich. System Grokking – A Novel Approach for Software Understanding, Validation, and Evolution, Lecture Notes in Computer Science, 2009, Volume 5831/2009, 38–49